Towards Self-Validating Knowledge-Based Archives*

Bertram Ludäscher Richard Marciano Reagan Moore

San Diego Supercomputer Center, U.C. San Diego

{ludaesch,marciano,moore}@sdsc.edu

Abstract

Digital archives are dedicated to the long-term preservation of electronic information and have the mandate to enable sustained access despite a rapidly changing information infrastructure. Current archival approaches build upon standardized data formats and simple metadata mechanisms for collection management, but do not *involve high-level conceptual models and knowledge representations. This results in serious limitations, not only for expressing various kinds of information and knowledge about the archived data, but also for creating infrastructure independent, self-validating and self-instantiating archives.*

To overcome these limitations, we first propose a scalable XML-based archival infrastructure, based on standard tools, and subsequently show how this architecture can be extended to a model-based framework, *where higher-level knowledge representations become an integral part of the archive and the ingestion/migration processes. This allows us to maximize infrastructure independence by archiving generic,* executable specifications *of (i) archival constraints (i.e., "model validators"), and (ii) archival transformations that are part of the ingestion process. The proposed architecture facilitates construction of* self-validating *and* self-instantiating *knowledge-based archives. We illustrate our overall approach and report on first experiences using a sample collection from a collaboration with the National Archives and Records Administration (NARA).*

1 Background and Overview

Digital libraries and archives, like their traditional paper-based counterparts, preserve data, information, and knowledge and thus are our "cultural memories" for future generations. While the rapidly evolving information technology provides ever-changing new opportunities for storing, managing, and accessing information, the plethora, complexity, and often short life-cycle of storage media, data formats, hardware, and software environments, all contribute to a serious challenge for the long-term preservation of information. Among other findings, the Task Force on Archiving Digital Information concluded that an infrastructure is needed that supports distributed systems of digital archives, and identified *data migration* as a crucial means for the sustained access to digital information [4].

In a research collaboration with the National Archives and Records Administration (NARA), the San Diego Supercomputer Center (SDSC) developed an information management architecture and prototype for digital archives, based on scalable archival storage systems (HPSS), data handling middleware (SRB/MCAT), and XML-based mediation techniques (MIX) [7, 10, 1].[1]

A core problem for persistent digital archives is the preservation of data collections in such a way that a faithful representation of their content can be dynamically reinstantiated in the future. To meet this goal, it is not sufficient to merely migrate data at the physically level from obsolete to current media but to create "recoverable" archival representations that are *infrastructure independent* (or *generic*) to the largest extent possible. Indeed the challenge is the forward-migration in time of *information and knowledge about* the archived data, *i.e.*, of the various kinds of meta-information that will allow recreation and interpretation of structure and content of archived data.

In this paper, we develop an architecture for infrastructure independent, *model-based* archival and collection management. Our approach is model-based in the sense that the ingestion process can employ both structural and semantic models of the collection, including a "flattened" relational representation, a "reassembled" semistructured representation, and higher-level "semantic"

*This research has been sponsored by the National Archives and Records Administration and Advanced Research Projects Agency/ITO, "Intelligent Metacomputing Testbed", ARPA Order No. D570, issued by ESC/ENS under Contract #F19628-96-C-0020

[1] `www.clearlake.ibm.com/hpss/`, `www.npaci.edu/DICE/SRB`, and `www.npaci.edu/DICE/MIX/`

representation. The architecture is *modular* since the ingestion process consists of transformations that are put together and executed in a *pipelined* fashion. Another novel feature of our approach is that we allow *archiving of the entire ingestion pipeline*, *i.e.*, the different representations of the collection together with the transformation rules that were used to create those representations.

The organization of the paper is as follows: In Section 2, we present the elements of a fully XML-based archival infrastructure. In Section 3 we show how this architecture can be extended to further include conceptual-level information and knowledge about the archived information. A unified perspective on XML-based "semantic extensions" is provided by viewing them as *constraint languages*. Most importantly, the notions of *self-validating* (Section 3.2) and *self-instantiating* (Section 3.3) archives are given precise meanings based on our formalization of the ingestion process. We report on first experiences using a real-world collection in Section 4 and conclude in Section 5.

2 XML-Based Digital Archives

In this section, we outline the architecture of an XML-based archive, using basic concepts and terminology from the recent reference model for an Open Archival Information System (OAIS) [9]. In Section 3 we propose an extension to this architecture by incorporating higher-level information (conceptual models and constraints) into the archival process.

Archival Processes and Functions. The primary goals of a digital archive are long-term information preservation together with the ability for sustained access of the archived information for later dissemination (Fig. 1): Initially, the information producer and the archive need to agree on the *submission* or *accessioning policies* (*e.g.*, acceptable submission formats, specifications on what is to be preserved, access functions, and other legal requirements). Subsequently, the producer can transfer *submission information packages* (SIPs) to the archive, where they enter – in our proposed system – an *ingestion network* (cf. Definition 3). An initial *quality assurance* check is performed on SIPs and corresponding feedback returned to the producer. As the SIPs are transformed within the ingestion network, *archival information packages* (AIPs) are produced and put into *archival storage*.

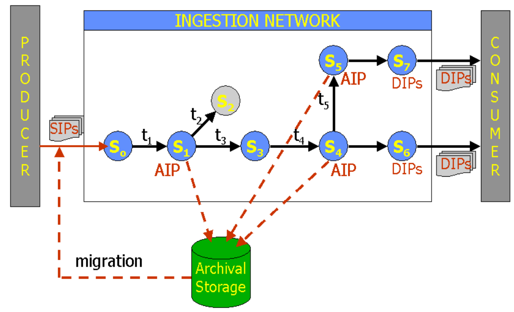

Figure 1. Digital archive architecture

Migration of AIPs is a normal "refreshing" operation of archives to prevent the obsolescence of AIPs in the presence of changes in technology or the contextual information that is necessary to understand the AIPs. Indeed, failure to migrate AIPs in a timely manner jeopardizes the creation of *dissemination information packages* (DIPs) at a later time.[2] Migration may be seen as a feedback of the AIPs into an *updated* ingestion network with the goal to preserve the full information content (in this sense, the initial ingestion can be viewed as the 0^{th} migration round). Creation of DIPs from AIPs is sometimes called *(re)-instantiation* of the archived collection.

The ingestion and migration *transformations* can be of different nature and involve data *reformatting* (*e.g.*, physical: from 6250 to 3480 tape, or bit-level: from EBCDIC to ASCII), and data *conversion* (*e.g.*"`.rtf`" to "`.html`") Conversions have to be content-preserving and should also preserve as much structure as possible. However, sometimes content is "buried" in the structure and thus can be lost accidentally during an apparently "content-preserving" conversion (Section 4). Finally, for dissemination and high-level "intelligent access", further transformations can be applied, *e.g.*, to derive a *topic map* view [12] on the data. Documentation about the sequence of transformations that have been applied to an AIP has to be provided and added to the *provenance* meta-information.

[2]Hypothetical access failures could be paraphrased as *"no such tape reader"*, *"no such access software"*, or *"no such expert"*. The last situation can occur when SIPs can be understood only by the expert data producer, but are not self-contained enough to be understood by a non-expert: *e.g.*, without the contextual information provided by the ingestion net "Rosetta Stone", information buried in the "hieroglyphic AIPs" could be lost forever.

Information Packages. In order to understand the disseminated information, the consumer (Fig. 1) not only needs some initial, internal knowledge (*e.g.*, of the English language), but also explicit *representation information* which is packed together with the actual content. Intuitively, the more representation information that is added to a package, the more self-contained it becomes. According to the OAIS framework [9], an *information package* IP contains *packaging information* PI (*e.g.*, the ISO-9660 directory information of a CD) that encapsulates the actual *content information* CI and additional *preservation description information* PDI. The latter holds information about the associated CI's *provenance* PR (origin and processing history), *context* CON (relation to information external to the IP), *reference* REF (for identifying the CI, say via ISBN or URI), and *fixity* information FIX (*e.g.*, a checksum over CI). Finally, similar to a real tag on a physical object, the IP has *descriptive information* DI on the "outside" that is used to discover which IP has the CI of interest. Put together, the encapsulation structure of an IP is as follows [9]:

- IP = [DI [PI [CI PDI[PR CON REF FIX]]]] (*)

Information Hierarchy. Information contained in an archive or IP can be classified as follows: At the *data*- or *instance-level*, there are individual digital objects like tuples and records. Such object-level information is packaged into the CI. At the *schema*- or *class-level*, structural and type information is handled: this metadata describes types of object attributes, aggregation information (collections/subcollections), and further descriptive *collection-level metadata*. For example, SDSC's Storage Resource Broker (SRB/MCAT)[3] provides a state-of-the-art "collection-aware" archival infrastructure. Collection-level metadata can be put into the PI and the DI. Finally, information at the *conceptual-level* captures *knowledge* about the archive and includes, *e.g.*, associations between concepts and object classes, relationships between concepts, and derived knowledge (expressed via logic rules). While some of this knowledge fits into the CON package, we suggest to provide a distinct *knowledge package* KP as part of the PDI. Possible representation formalisms for expressing such knowledge range from database-related formalisms like (E)ER diagrams, UML class diagrams, and XML Schema, to more AI/KR-related formalisms like RDF(-Schema), semantic networks, ontologies, description logics, etc. Clearly, for archival purposes, robust and (close to) standard formalism like XMI (which includes UML model exchange and OMG's Meta Object Facility) [13], RDF [11], the Conceptual Graph Standard [2] and the Knowledge-Interchange Format [5] are candidate archival formats. However the number of possible formalisms (and the complexity of some of them) makes this a daunting task for any archival system. A better solution is to employ a generic, universal formalism which can express all of the above via executable specifications (see below).

[3] `www.npaci.edu/DICE/SRB`

XML-Based Archival Infrastructure. It is desirable that archival formats do not require special access software and be *standardized*, *open*, and as *simple* as possible. Ideally, they should be "self-contained" and "self-describing". The specifications of proprietary formats[4] like Word, Wordperfect, etc. may vary from version to version, may not be available at all, or — even if they are available (e.g., RTF) — may still require custom software ("viewers") to "understand" the information contained in a document. Similary, formats that use data compression like PDF require that the specification describes the compression method and that this method is executable in the future. XML, on the other hand, satisfies many desiderata of archival: the language is standardized [14], and easy to understand (by humans) and parse (by programs). Document structure and semantics can be encoded via user-definable tags (*markup*), sometimes called *semantic tags*, since they facilitate *separation of content from presentation* (unlike HTML which mixes them). Because of user-defined tags, XML can be seen as a *generic, self-describing* data format.

Viewed as a data model, XML corresponds to *labeled, ordered trees*, *i.e.*, a *semistructured data model*. Consequently, XML can easily express the whole range from highly structured information (records, database tables, object structures) to very loosely structured information (HTML, free text with some markup). In particular, the structure of an information package IP as indicated in (*) can be directly represented with *XML elements*: IPs (and contained sub-IPs) are *encapsulated* via

[4] Proprietary formats like "`.doc`" tend to be complex, undocumented, and "married" to a hardware or software environment. Data formats whose specifications can be "grasped easily" (both physically and intellectually) and for which tools-support is available, are good candidate archival formats.

delimiting opening and closing tags; descriptive (meta)-information DI about a package can be attached in XML *attributes*, etc. XML elements can be *nested* and – since order of subelements is preserved – ordered and unordered collection types (*list*, *bag*, *set*) can be easily encoded, thereby directly supporting collection-based archives.

The core of our archival architecture is the ingestion network. Some distinguished nodes (or *stages*) of the ingestion net produce AIPs, others yield different "external views" (DIPs). As IPs pass from one stage to the next, they are queried and restructured like database instances. At the syntactic level, one can maximize infrastructure independence by representing the databases in XML and employing *standard tools* for parsing, querying, transforming, and presenting XML.[5] To ensure *modularity* of the architecture, complex XML transformations should be broken up into smaller ones that can be expressed directly with the available tools. For supporting *huge data volumes* and *continuous streams* of IPs, the architecture needs to be *scalable*. This can be achieved with a *pipelining execution model* using *stream-based* XML transformation languages (*i.e.*, whose memory requirements do *not* depend on the size of the XML "sent over the wire"). As the XML IPs are being transformed in the ingestion net, provenance information PR is added. This includes the usual identification of the organizational unit and individuals who performed the migration, as well as identification of the *sequence of XML mappings* that was applied to the IP. By storing *executable specifications* of these mappings, *self-instantiating* archives can be built (Section 3.3).

3 XML vs. Model-Based Archival

In this section, we propose to extend the purely structural approach of plain XML to include more *semantic information*. By employing "executable" knowledge representation formalisms, one can not only capture more semantics of the archived collection, but this additional information can also be used to *automatically validate* archives at a higher, conceptual level than before where it was limited to low-level fixity information or simple structural checks.

Intuitively, we speak of a *model-based* or *knowledge-based* archival approach, if IPs can contain conceptual-level information in *knowledge packages* (KPs). The most important reason to include KPs is that they capture meta-information that may otherwise be lost: For example, at ingestion time it may be known that digital objects of one class inherit certain properties from a superclass, or that functional or other dependencies exists between attributes, etc. Unfortunately, more often than not, such valuable information is not archived explicitly.

During the overall archival process, KPs also provide additional opportunities and means for quality assurance: At ingestion time, KPs can be used to check that SIPs indeed conform to the given accessioning policies and corresponding feedback can be given to the producer. During archival management, *i.e.*, at migration or dissemination time, KPs can be used to *verify* that the CI satisfies the pre-specified *integrity constraints* implied by the KPs. Such value-added functions are traditionally not considered part of an archival organization's responsibilities. On the other hand, the detection of "higher-level inconsistencies" clearly yields valuable meta-information for the producers and consumers of the archived information and could become an integral service of future archives.

The current approach for "fixing the meaning" of a data exchange/archival format is to provide an XML DTD. For example, many organizations and groups defined their "community language" in this way. However, the fact that a document has been *validated* say wrt. the Encoded Archival Description DTD [3] does *not* imply that it satisfies all constraints that are part of the EAD specification. Indeed, only *structural constraints* can be automatically checked using a (DTD-based) validating parser – all other constraints are not checked at all or require specialized software.

These and other shortcomings of DTDs for data modeling and validation have been widely recognized and have led to a flood of extensions, ranging from the heavyweight, W3C-supported XML Schema proposal [15],[6] to more grassroots efforts like `relax` (which may become a standard) [8],[7] and many others (RDF, RDF-Schema, SOX, DSD, Schematron, XML-Data, DCD, XSchema/DDML, ...). A unifying perspective on these languages can be achieved by viewing them as *constraint languages* that distinguish "good documents" (those that are *valid* wrt. the constraints) from "bad" (*invalid*) ones.

[5] *e.g.*, SAX, XPath, Quilt, XSLT, ...

[6] ≈ DTDs + datatypes + type extensions/restrictions + ...

[7] ≈ DTDs + (datatypes, ancestor-sensitive content models, local scoping, ...) – (entities, notations) ...

3.1 XML Extensions as Constraint Languages

Assume IPs are expressed in some *archival language* $\mathcal{A}$. In the sequel, let $\mathcal{A} \supseteq$ XML. A concrete *archive instance* (short: *archive*) is a "word" a of the archival language $\mathcal{A}$, *e.g.*, an XML document.

Definition 1 (Archival Constraint Languages) We say that $\mathcal{C}$ is a *constraint language* for $\mathcal{A}$, if for all $\varphi \in \mathcal{C}$ the set $\mathcal{V}_\varphi = \{a \in \mathcal{A} \mid a \models \varphi\}$ of *valid archives* (wrt. φ) is decidable. □

For example, for $\mathcal{C} =$ DTD, a *constraint* φ is a concrete DTD: for any document $a \in$ XML, validity of a wrt. the DTD φ is decidable (any so-called "validating XML parser" checks whether $a \models \varphi$). The notion of constraint language provides a unifying perspective and the basis for comparing formalisms like DTD, XML-SCHEMA, RELAX, RDF-SCHEMA, wrt. their expressiveness and complexity.

Definition 2 (Subsumption) We say that $\mathcal{C}'$ *subsumes* $\mathcal{C}$ *wrt.* $\mathcal{A}$, denoted $\mathcal{C}' \succ \mathcal{C}$, if for all $\varphi \in \mathcal{C}$ there is a $enc(\varphi) \in \mathcal{C}'$ s.t. for all $a \in \mathcal{A}$: $a \models \varphi$ iff $a \models enc(\varphi)$. □

As a constraint language, DTD can express only certain structural constraints over XML, all of which have equivalent encodings in XML-SCHEMA. Hence XML-SCHEMA subsumes DTD. On the other hand, XML-SCHEMA is a much more complex formalism than DTD, so a more complex validator is needed when reinstantiating the archive, thereby actually increasing the infrastructure dependence (at least for archives where DTD constraints are sufficient). To overcome this problem, we propose to use a generic, universal formalism that allows one to *specify and execute* other constraint languages:

3.2 Self-Validating Archives

Example 1 (Logic DTD Validator) Consider the following F-LOGIC rules [6]:

```
%%% Rules for ⟨!ELEMENT X    (Y,Z) ⟩
(1) false ←P:X, not (P.1):Y.
(2) false ←P:X, not (P.2):Z.
(3) false ←P:X, not P[_→_].
(4) false ←P:X[N→_], not N=1, not N=2.
%%% Rules for ⟨!ELEMENT X    (Y | Z) ⟩
(5) false ←P:X[1→A], not A:Y, not A:Z
(6) false ←P:X, not P[_→_].
(7) false ←P:X[N→_], not N=1.
%%% Rule for ⟨!ELEMENT X    (Y)* ⟩
(8) false ←P:X[_→C], not C:Y.
```

The rule templates illustrate how to generate for each $\varphi \in$ DTD a logic program $enc(\varphi)$ in F-LOGIC, which derives `false` iff a given document $a \in$ XML is not valid wrt. φ: e.g., if the first child is not `Y` (1), or if there are more than two children (4). □

The previous logical DTD specification does not involve recursion and can be expressed in classical first-order logic FO. However, for expressing transitive constraints (*e.g.*, subclassing, value inheritance, etc.) fixpoint extensions to FO (like DATALOG or F-LOGIC) are necessary.

Proposition 1 *(i)* XML-SCHEMA $\succ$ DTD, *(ii)* F-LOGIC $\succ$ DTD. □

Note that there is a subtle but important difference between the two subsumptions: In order to "recover" the original DTD constraint via *(i)*, one needs to understand the specific XML-SCHEMA standard, and in order to execute (*i.e.*, check) the constraint, one needs a *specific* XML-SCHEMA validator. In contrast, the subsumption of *(ii)* as sketched above contains its own *declarative, executable specification*, hence is *self-contained* and *infrastructure independent*. In this case, *i.e.*, if an AIP contains (in KP) an *executable specification* of the constraint φ, we speak of a *self-validating archive*. This means that at dissemination time we only need a single *generic logic engine* (*e.g.*, to execute F-LOGIC or PROLOG) on which we can run all logically defined constraints. The generic engine for executing "foreign constraints" does not have to be a logic one though: *e.g.*, a RELAX validator has been written in XSLT [16]. Then, at reinstantiation time, one only needs a generic XSLT engine for checking RELAX constraints.[8]

3.3 Self-Instantiating Archives

A self-validating archive captures one or more snapshots of the archived collection at certain stages during the ingestion process, together with constraints φ for each snapshot. The notion of *self-instantiating archive* goes a step further and aims at archiving also the *transformations* of the ingestion network themselves. Thus, instead of adding only descriptive metadata about a transformation which is external to the archive, we include the "transformation knowledge" thereby internalizing complete parts of the ingestion process.

[8]However, in the archival context, instead of employing the latest, rapidly changing formalisms, a "timeless" logical approach may be preferable.

As before, we can maximize infrastructure independence by employing a universal formalism whose specifications can be executed on a virtual (logic or XML-based) engine – ideally the same one as used for checking constraints. To do so, we model an ingestion network as a graph of *database transformations*. This is a natural assumption for most real transformations (apart form very low level reformatting and conversion steps).

Definition 3 (Ingestion Network) Let $\mathcal{T}$ be a set of transformations $t : \mathcal{A} \rightarrow \mathcal{A}$, and $\mathcal{S}$ a set of *stages*. An *ingestion network* $\mathcal{IN}$ is a finite set of labeled edges $s \rightarrow_t s'$, having associated *preconditions* $\varphi(s)$ and *postconditions* $\varphi(s')$, for $s, s' \in \mathcal{S}, t \in \mathcal{T}, \varphi(s), \varphi(s') \in \mathcal{C}$. □

We call the edges of $\mathcal{IN}$ *pipes* and say that an archive $a \in \mathcal{A}$ is *acceptable for* ("may pass through") the pipe $s \rightarrow_t s'$, if $a \models \varphi(s)$ and $t(a) \models \varphi(s')$. Since $\mathcal{IN}$ can have loops, fixpoint or closure operations can be handled. If there are multiple t-edges $s \rightarrow_t s'_i$ outgoing from s, then one s'_0 is distinguished to identify the *main pipe* $s \rightarrow_t s'_0$; the remaining $s \rightarrow_t s'_i$ are called *contingency pipes*. The idea is that the postcondition $\varphi(s'_0)$ captures the normal, desired case for applying t at s, whereas the other $\varphi(s'_i)$ handle exceptions and errors. In particular, for $\varphi(s'_1) = \neg\,\varphi(s'_0)$ we catch *all* archives that fail the main pipe at s, so s'_1 can be used to abort the ingestion and report the integrity violation $\neg\,\varphi(s'_0)$. Alternatively, s'_1 may have further outgoing contingency pipes aimed at rectifying the problem.

When an archive a successfully passes through the ingestion net, one or more of the transformed versions a' are archived. One benefit of archiving the transformations of the pipeline (SIP $\rightarrow_{t_1} \cdots \rightarrow_{t_n}$ AIP) in an infrastructure independent way is that knowledge, that was available at ingestion time and is possibly hidden within the transformation, is preserved. Moreover, some of the transformations yield user-views (AIP $\rightarrow_{t_1} \cdots \rightarrow_{t_m}$ DIP), *e.g.*, topic maps or HTML pages. By archiving self-contained, executable specifications of these mappings, the archival reinstantiation process can be automated to a large extent using infrastructure independent representations.

Properties of Transformations. Finally, by modeling the ingestion net as a graph of database mappings, we can formally study properties of the ingestion process, *e.g.*, the data complexity of a transformation or whether the transformation is invertible or not. Note that invertible mappings are content preserving. For transformations t that are not specific to a collection, it can be worthwile to derive and implement the inverse mapping t^{-1} thereby guaranteeing that t is content preserving.

Example 2 (Inverse Wrapper) Consider a document collection $\{a_1, a_2, \ldots\} \subseteq$ HTML for which a common wrapper t has been provided s.t. $t(a_i) = a'_i \in$ XML.[9] The exact inverse mapping may be impracticable to construct, but a "reasonably equivalent" t^{-1} (*i.e.*, modulo whitespaces, irrelevant formatting details, etc.) may be easy to define as an XSLT stylesheet. Thus, the output of the pipe $a_i \rightarrow_t a'_i \rightarrow_{t^{-1}} a''_i \in$ HTML can be seen as a *normalized* HTML version of the input a_i. By restricting to such normalized input, t becomes invertible, and the XSLT acts as an "inverse wrapper" for presenting the collection. □

4 Case Study: The Senate Collection

In a research collaboration with the National Archives and Records Administration (NARA), SDSC developed an information management architecture and prototype for digital archives. In the sequel, we illustrate some of the aspects of our archival architecture, using the *Senate Legislative Activities* collection (SLA), one of several reference collections that NARA provided for research purposes.

Collection Submission and Initial Model. The SLA collection contains an extract of the 106th Congress database *bills*, *amendments*, and *resolutions* (short: BARs). SLA was physically submitted on CD-ROM as 99 files in Microsoft's *Rich Text Format* (RTF), one per *active* senator, and organized to reflect a particular senator's legislative contribution over the course of the 106th Congress. Based on a visual inspection of the files, an *initial conceptual model* CM_0 with the following structure was assumed:

- **Header section**: includes the *senator name* (*e.g.*, "`Paul S. Sarbanes`"), *state* ("`Maryland`"), *reporting period* ("`January 06, 1999 to March 31, 2000`"), and *reporting entity* ("`Senate Computer Center Office of the Sergeant at Arms and Committee on Rules and Administration`")
- **Section I**: *Sponsored Measures*, **Section II**: *Cosponsored Measures*, **Section III**: *Sponsored Measures Grouped by Committee Referral*, **Section IV**:

[9] In this example, the archival language must include both sublanguages, *i.e.*, $\mathcal{A} \supseteq$ HTML $\cup$ XML.

Cosponsored Measures Organized by Committee Referral, **Section V**: *Sponsored Amendments*, **Section VI**: *Cosponsored Amendments*,

- **Section VII**: *Subject Index to Sponsored and Cosponsored Measures and Amendments.*

CM_0 also modeled the fact that Sections III and IV contain the same bills and amendments as Sections I and II, but *grouped by committee referral* (*e.g.,* "`Senate Armed Services`" and "`House Judiciary`"), and that Section VII contains a list of subjects with references to corresponding BAR indentifiers: "`Zoning and zoning law` $\rightarrow$ `S.9, S.Con.Res.10, S.Res.41, S.J.Res.39`". *Measures* are *bills* and *resolutions*; the latter have three subtypes: *simple*, *joint*, and *concurrent*.

Finally, CM_0 identified 14 initial *data fields* DF_0 (=attributes) that needed to be extracted.[10]

Ingestion Process. Figure 2 depicts the ingestion network as it eventually evolved: The (presumed) conversion from (MS Word) DOC to RTF happened outside of the ingestion net, since the accessioning policy prescribed SIPs in RTF format.

- $S_1 \rightarrow S_2$:[11] A first, supposedly content preserving, conversion to HTML using MS Word turned out to be lossy when checked against CM_0: the groupings in Sections III and IV were no longer part of the HTML files,[12] so it was impossible to associate a measure with a committee!
- $S_1 \rightarrow S_3$: the conversion from RTF to an information preserving XML representation was accomplished using an `rtf2xml` module[13] for OmniMark, a stream-oriented rule-based data extraction and programming language.
- $S_3 \rightarrow S_4$: this main wrapping step was used to extract data according to the initial data fields DF_0. In order to simplify the (Perl) wrapper module and make it more generic, we used a *flat, occurrence-based* representation for data extraction: each data field (attribute) was recorded in OAV form, *i.e.*,

(*occurrence*, *attribute*, *value*)

The occurrence has to be fine-grained enough for the transformation to be information preserving (in our case *occurrence* = (*filename*, *line-number*)). The *scope of an occurrence* is that part of the linearized document which defines the extent of the occurrence. For example, in case of an occurrence based on line numbers, the scope is from the first character of the line to the last character of the line. In case of XML, the scope of an occurrence may often be associated with element boundaries (but finer occurrence granules may be defined for XML as well).

By employing the "deconstructing" OAV model, the wrapper program could be designed simpler, more modular and thus easier to reuse. For example, `date_introduced` could show up in the file of Senator Paul Sarbanes (`senator_id=106`) at line 25 with value `01/19/1999` and also in line 106 at line 55 with value `03/15/2000`. This information is recorded with two tuples: `((106,25), 'date_introduced', '01/19/1999')` and `((106,55), 'date_introduced', '03/15/2000')`.

- $S_4 \rightarrow S_4$: some candidate attributes from DF_0 had to be decomposed further, which is modeled by a recursive *closure step* $S_4 \rightarrow S_4$, corresponding to a sequence $DF_1, \ldots, DF_n$ of refinements of the data-fields, *e.g.*, DF_1: `list_of_sponsors` $\rightarrow$ `[sponsor]`, and DF_2: `sponsor`$\rightarrow$ `(name, date)`.
- $S_4 \rightarrow S_5$: this "reconstructing" step builds the desired archival information packages AIP in XML. Content *and structure* of the original SIPs is preserved by reassembling larger objects from subobjects using their occurrence values. From the created XML AIPs, DTDs like the following can be inferred (and included as a constraint φ in KP):

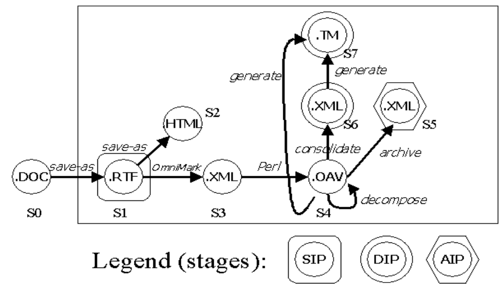

Figure 2. Ingestion Network: Senate Collection

[10] *abstract, bar_id, committee, congressional_record, cosponsors, date_introduced, digest, latest_status, official_title, sponsor, statement_of_purpose, status_actions, submitted_by, submitted_for*

[11] this dead end is only an example for existing pitfalls; $S_1 \rightarrow S_2$ is *not* archived.

[12] this crucial information was part of the RTF *page header* but left no trace whatsoever in the HTML

[13] from Rick Geimer at `xmeta.com`

```
<!ELEMENT SLA_collection
      (senate_file*)>
<!ELEMENT senate_file
      (file_name, header_page?,
      section*, subject_index?)>
<!ELEMENT section
      (sec_number, sec_name, bar*)>
<!ELEMENT bar
      (bill | amendment | resolution)>
...
```

• $S_4 \rightarrow S_6$: this conceptual-level transformation creates a *consolidated version* from the collection. For example, SLA contains 44,145 *occurrences* of BARs, however there are only 5,632 *distinct* BAR objects. (Alternatively, this version could have been derived from S_5.) This step can be seen as a reverse-engineering of the original database content, of which SLA is only a view (group BARs by senator, for each senator group by measures, committee, etc.)

As part of the consolidation transformation, it is natural to perform conceptual-level integrity checks: *e.g.*, at this level it is easy to define a constraint φ that checks for *completeness* of the collection (*i.e.*, if each senator occurring somewhere in the collection also has a corresponding senator file – a simple declarative query reveals the answer: no!). Note that a consolidated version provides an *additional* archival service; but it is mandatory to also preserve a non-consolidated "raw version" (e.g., as derived from the OAV model).

• $S_4, S_6 \rightarrow S_7$: these transformations create a *topic map* version and thus provide additional conceptual-level "hooks" into the consolidated and OAV version.

5 Conclusions

We have presented an archival infrastructure for self-validating knowledge-based archives: *self-validating* means that declarative constraints about the collection are included in *executable form* (as logic rules). Most parts of the ingestion network (apart from S_7 which is under development) have been implemented for a concrete collection. Note that all transformations following the OAV format can be very naturally expressed in a high-level object-oriented query language like F-Logic. By including the corresponding rules as part of the archive, a *self-instantiating*, self-validating archive can be constructed. In future work we will consider two specific problems of the ingestion process, *i.e.*, *closure* (selecting the "right set" of attributes for CM_0 and OAV) and *completeness* (all attributes are populated).

References

[1] C. Baru, V. Chu, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. XML-Based Information Mediation for Digital Libraries. In *ACM Conf. on Digital Libraries*, pages 214–215, Berkeley, CA, 1999. exhibition program.

[2] Conceptual Graph Standard. dpANS, NCITS.T2/98-003, `http://www.bestweb.net/~sowa/cg/cgdpans.htm`, Aug. 1999.

[3] Encoded Archival Description (EAD). `http://lcweb.loc.gov/ead/`, June 1998.

[4] J. Garrett and D. Waters, editors. *Preserving Digital Information – Report of the Task Force on Archiving of Digital Information*, May 1996. `http://www.rlg.org/ArchTF/`.

[5] Knowledge Interchange Format (KIF). dpANS, NCITS.T2/98-004, `http://logic.stanford.edu/kif/`, 1999.

[6] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[7] R. Moore, C. Baru, A. Rajasekar, B. Ludäscher, R. Marciano, M. Wan, W. Schroeder, and A. Gupta. Collection-Based Persistent Digital Archives. *D-Lib Magazine*, 6(3,4), 2000.

[8] M. Murata. RELAX (REgular LAnguage description for XML). `http://www.xml.gr.jp/relax/`, Oct. 2000.

[9] Reference Model for an Open Archival Information System (OAIS). submitted as ISO draft, `http://www.ccsds.org/documents/pdf/CCSDS-650.0-R-1.pdf`, 1999.

[10] A. Paepcke, R. Brandriff, G. Janee, R. Larson, B. Ludäscher, S. Melnik, and S. Raghavan. Search Middleware and the Simple Digital Library Interoperability Protocol. *D-Lib Magazine*, 6(3), 2000.

[11] Resource Description Framework (RDF). W3C Recommendation `www.w3.org/TR/REC-rdf-syntax`, Feb. 1999.

[12] ISO/IEC FCD 13250 – Topic Maps, 1999. `http://www.ornl.gov/sgml/sc34/document/0058.htm`.

[13] OMG XML Metadata Interchange (XMI). `www.omg.org/cgi-bin/doc?ad/99-10-02`, 1999.

[14] Extensible Markup Language (XML). `www.w3.org/XML/`, 1998.

[15] XML Schema, Working Draft. `www.w3.org/TR/xmlschema-{1,2}`, Sept. 2000.

[16] K. Yonekura. RELAX Verifier for XSLT. `http://www.geocities.co.jp/SiliconValley-Bay/4639/`, Oct. 2000.